INF8775 – Analyse et conception d’algorithmes

TP1 – Automne 2023

|  |  |
| --- | --- |
| **Nom, prénom, matricule des membres** | Berton Margot 2314078  Plazziac Thomas 2317184 |
| **Note finale / 13** | 0 |

# Informations techniques

* Répondez directement dans ce document .docx. Veuillez ne pas inclure le texte en italique servant de directive. La correction se fait sur ce même rapport.
* Vous devez faire une remise électronique sur Moodle avant le 10 octobre à 23h55 pour le groupe 1, 3 octobre à 23h55 pour le groupe 2 et 5 octobre à 23h55 pour le groupe 3.
* Vos fichiers doivent être remis dans une archive zip à la racine de laquelle on retrouve:
  + Ce rapport sous format docx.
  + Un script nommé tp.sh servant à exécuter les différents algorithmes du TP. L’interface du script est décrite à la fin du rapport.
  + Le code source et les exécutables.
  + Vous avez le choix du langage de programmation utilisé mais vous devrez utiliser les mêmes langage, compilateur et ordinateur pour toutes vos implantations. Notez que le code et les exécutables soumis seront testés sur les ordinateurs de la salle L-4714 et doivent être compatibles avec cet environnement. En d’autres mots, tout doit fonctionner correctement lorsque le correcteur exécute votre script tp.sh sur un des ordinateurs de la salle.
* Si vous utilisez des extraits de codes (programmes) trouvés sur Internet, vous devez en mentionner la source, sinon vous serez sanctionnés pour plagiat.
* On vous encourage à lire le guide intitulé « guide bash » sur Moodle pour faire vos graphiques. C’est un guide qui a été conçu pour un ancien TP, mais il contient beaucoup d’informations utiles.

# Mise en situation

Ce travail pratique se répartit sur deux séances de laboratoire et porte sur l’analyse empirique et hybride des algorithmes. À la section 3.2 des notes de cours, trois approches d’analyse de l’implantation d’un algorithme sont décrites. Vous les mettrez en pratique pour des algorithmes de tri.

Vous implanterez les algorithmes de tri par dénombrement et de tri rapide (quicksort). Un rappel de ces algorithmes sera fait lors de la séance de laboratoire. Vous ferez trois versions de l'algorithme de tri rapide :

1. Pivot sur le 1er élément et seuil de récursivité à 1
2. Pivot sur le 1er élément et seuil de récursivité déterminé expérimentalement
3. Pivot sur un élément aléatoire et seuil de récursivité déterminé expérimentalement

Les exemplaires dont la taille est en deçà du seuil de récursivité ne sont plus résolus récursivement mais plutôt directement avec un algorithme de tri de votre choix prenant un temps dans Θ(n2) en pire cas (tri par insertion, par sélection, à bulles, etc.)

# Jeu de données

Vous trouverez dans l’archive du TP un script python *gen.py* servant à générer les exemplaires. Ce script s’exécute de la manière suivante :

gen.py -S TAILLE [-n NB\_EXEMPLAIRES] [-m MAGNITUDE] [-r RANDOM\_SEED] [-p PSEUDO-TRI]

TAILLE correspond à la taille des exemplaires.

NB\_EXEMPLAIRES correspond au nombre de suites que vous voulez générer pour chaque taille

MAGNITUDE correspond à la valeur maximale possible dans les exemplaires. Par exemple, pour -m 100000, les exemplaires comportent des nombres de 0 à 100000.

RANDOM\_SEED correspond à la *seed* utilisée pour la génération aléatoire des listes de nombres.

PSEUDO-TRI est une variable binaire qui détermine si l’exemplaire sera presque trié.

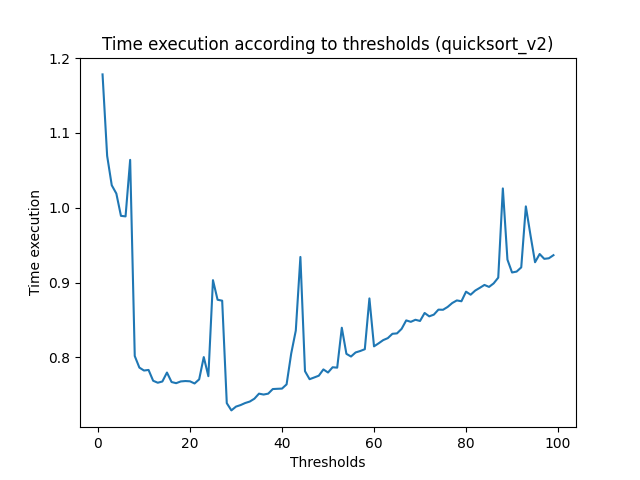
**Le script generate\_series.sh permet de réinitialiser le dossier samples avec nos séries.**

# Présentation des résultats

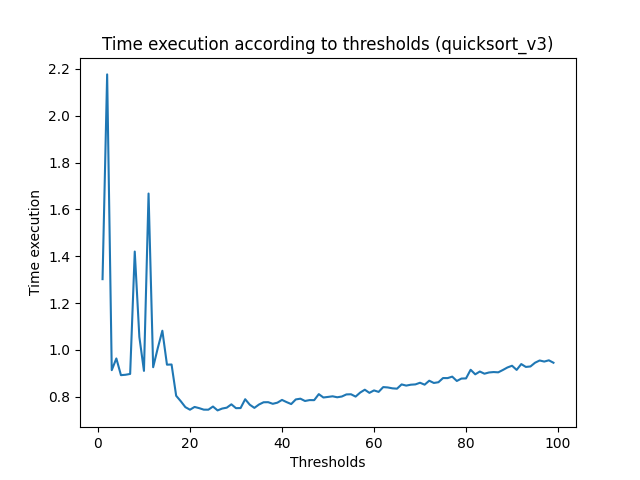
|  |  |
| --- | --- |
| 0 | / 4 pt |

### Tableau des résultats

Dans notre script find\_thresholds.py, nous cherchons empiriquement les seuils de récursivité pour quicksort\_v2 et quicksort\_v3. Nous avons tracé le temps d’exécution moyen (sur 10 listes de taille 400000 et de magnitude 1000000) en fonction du seuil :



*Fig. 1.a : Complexité en fonction du seuil de récursivité*



*Fig. 1.b : Complexité en fonction du seuil de récursivité*

Sur les figures ci-dessus, on lit un seuil de récursivité minimisant le temps d’exécution de quicksort\_v2 (resp. quicksort\_v3) égal à 29 (resp. 26). En répétant plusieurs fois l’algorithme, nous constatons que ces valeurs peuvent varier, mais se situent généralement entre 25 et 35. Nous avons mis 25 par défaut dans nos fonctions.

Ensuite, nous avons calculé les tableaux des résultats demandés :

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tri dénombrement** | | | | | | | | | | |
| **Série 1** | | |  | **Série 2** | | |  | **Série 3** | | |
| Taille | Magnitude | Temps (ms) | Taille | Magnitude | Temps (ms) | Taille | Magnitude | Temps (ms) |
| 50000 | 1000000 | 145.99 | 50000 | 500000 | 85.43 | 50000 | 500000 | 88.82 |
| 100000 | 199.29 | 1000000 | 141.28 | 100000 | 1000000 | 186.74 |
| 200000 | 268.05 | 2000000 | 279.83 | 200000 | 2000000 | 352.76 |
| 400000 | 383.12 | 4000000 | 691.00 | 400000 | 4000000 | 802.74 |

*Tab. 1 : Tri dénombrement*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Tri pivot v1 - seuil : 1, pivot : 1er élément** | | | | | | |
| **Série 1** | | |  | **Série 4** | | |
| Taille | Magnitude | Temps (ms) | Taille | Magnitude | Temps (ms) |
| 50000 | 1000000 | 81.90 | 50000 | 1000000 | 136.22 |
| 100000 | 181.71 | 100000 | 307.86 |
| 200000 | 416.41 | 200000 | 674.91 |
| 400000 | 1088.59 | 400000 | 1481.40 |

*Tab. 2 : Tri pivot v1*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Tri pivot v2 - seuil : 22, pivot : 1er élément** | | | | | | |
| **Série 1** | | |  | **Série 4** | | |
| Taille | Magnitude | Temps (ms) | Taille | Magnitude | Temps (ms) |
| 50000 | 1000000 | 66.45 | 50000 | 1000000 | 113.74 |
| 100000 | 141.97 | 100000 | 339.12 |
| 200000 | 326.44 | 200000 | 593.38 |
| 400000 | 768.64 | 400000 | 1371.91 |

*Tab. 3 : Tri pivot v2*

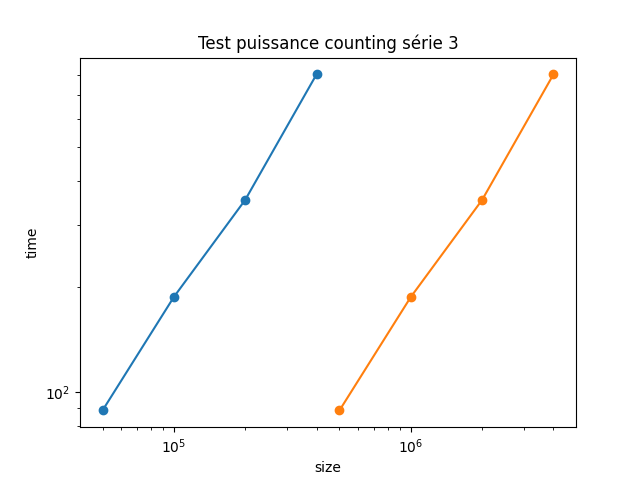
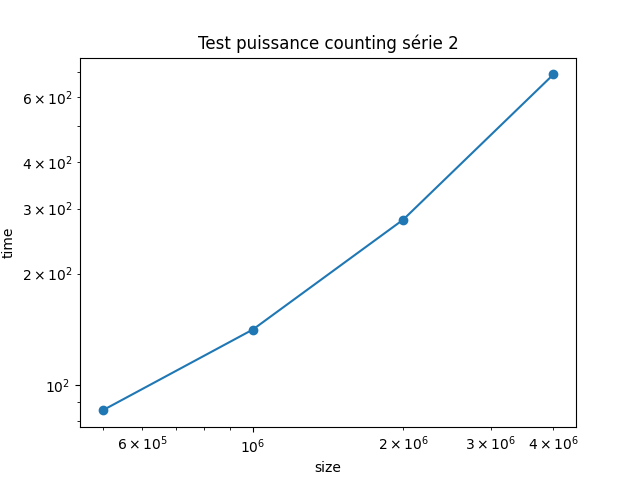
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Tri pivot v3 - seuil : 34, pivot : aléatoire** | | | | | | |
| **Série 1** | | |  | **Série 4** | | |
| Taille | Magnitude | Temps (ms) | Taille | Magnitude | Temps (ms) |
| 50000 | 1000000 | 70.93 | 50000 | 1000000 | 59.21 |
| 100000 | 150.68 | 100000 | 133.72 |
| 200000 | 369.08 | 200000 | 297.01 |
| 400000 | 770.22 | 400000 | 699.51 |

*Tab. 4 : Tri pivot v3*

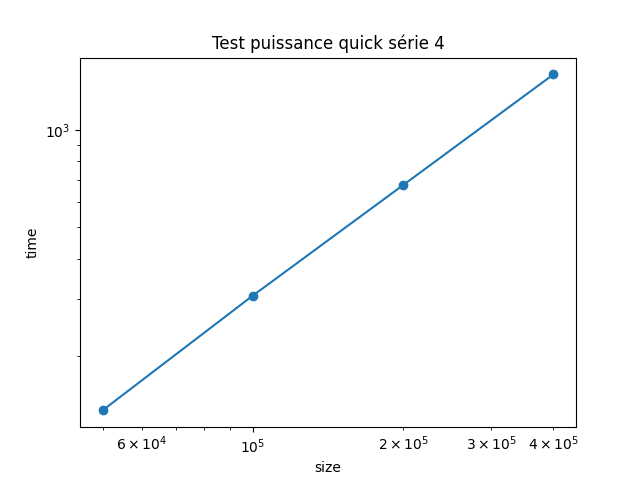
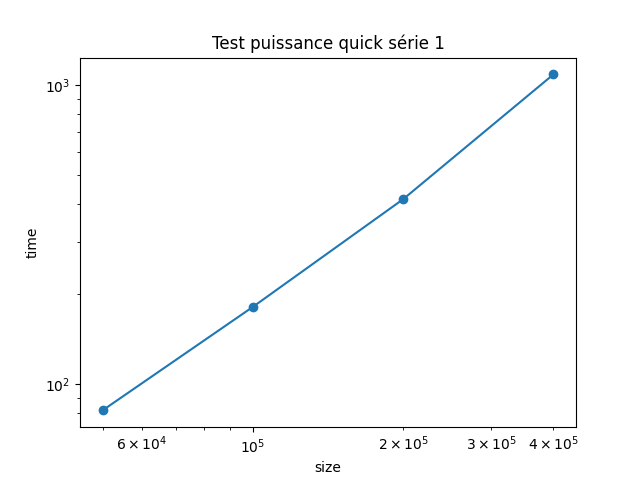
Quelques remarques à première vue :

* Pour les deux types de tri, augmenter la taille des exemplaires augmente le temps d’exécution du code. Pour le tri par dénombrement, augmenter la magnitude augmente également le temps d’exécution du code (ce qui est logique car ce tri comprend un tableau intermédiaire de taille égale à la magnitude).
* Pour une taille d’exemplaire faible, on privilégiera n’importe quelle version de tri rapide face au tri par dénombrement. En revanche, pour une taille d’exemplaire élevée, le tri par dénombrement devient plus avantageux.
* Pour n’importe quelle taille d’exemplaire fixée, utiliser un seuil de récursivité à 25 optimise les performances du tri rapide.
* Choisir un pivot aléatoire plutôt que le premier élément de la liste est assez avantageux. Cela ne fait pas de différence généralement… sauf si la liste est presque triée : dans ce cas-là, le temps d’exécution du code aurait été particulièrement long sinon.

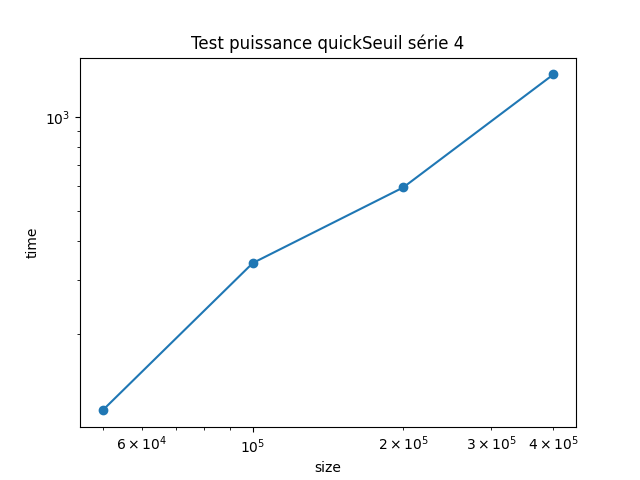
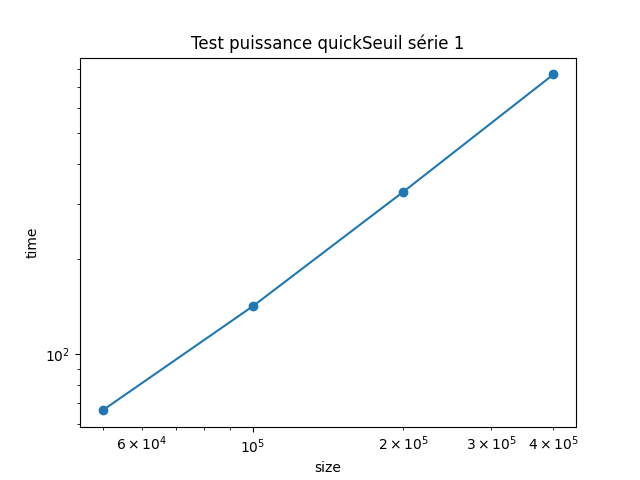
### Tests de puissance



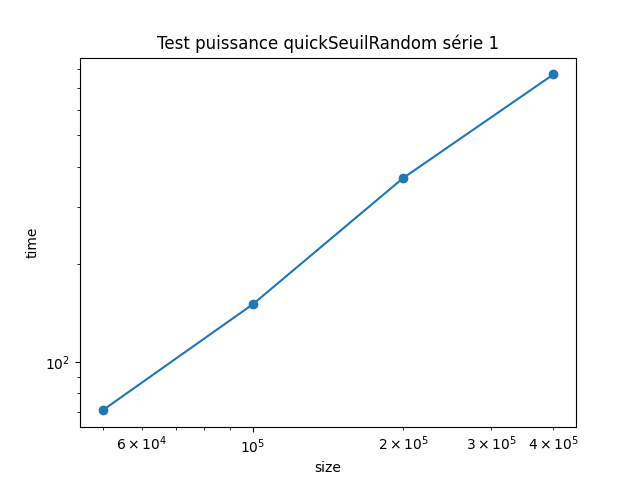
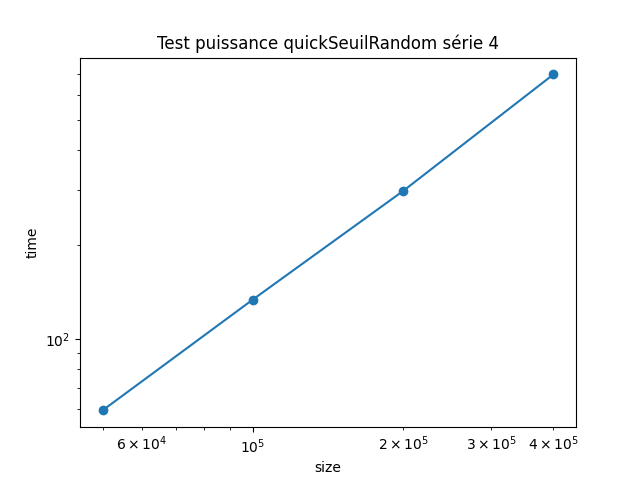
*Fig. 2.a, 2.b, 2.c (gauche à droite) : Tests de puissance pour tri par dénombrement*



*Fig. 3.a, 3.b (gauche à droite) : Tests de puissance pour tri rapide v1*

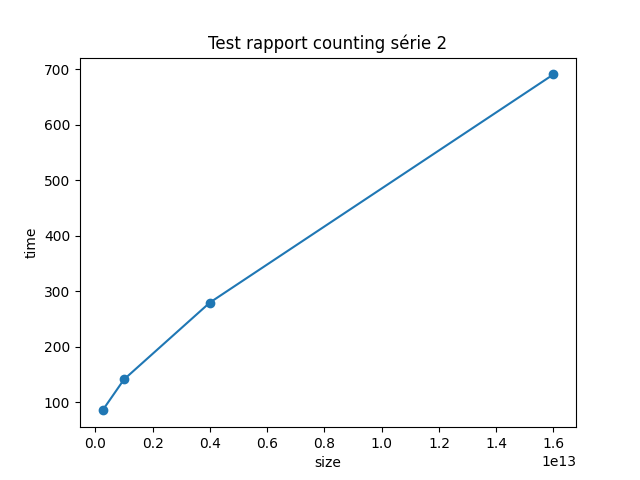


*Fig. 4.a, 4.b (gauche à droite) : Tests de puissance pour tri rapide v2*

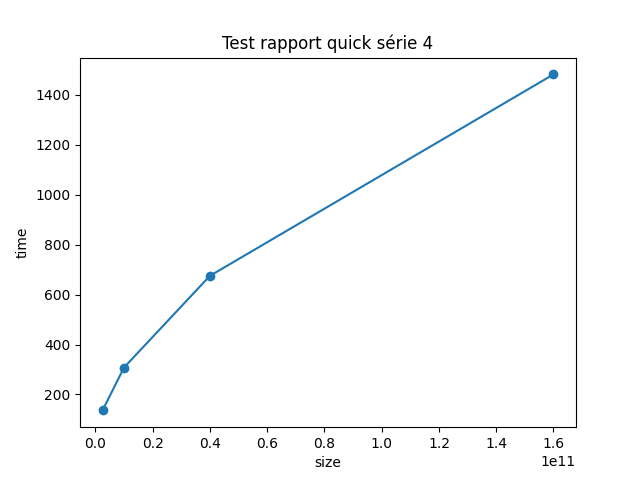
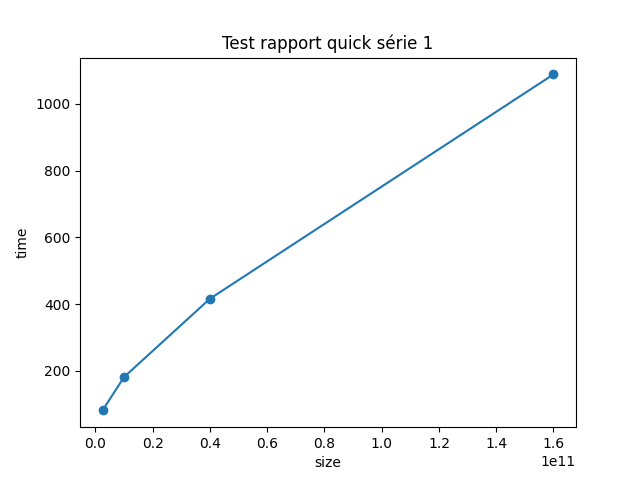


*Fig. 5.a, 5.b (gauche à droite) : Tests de puissance pour tri rapide v3*

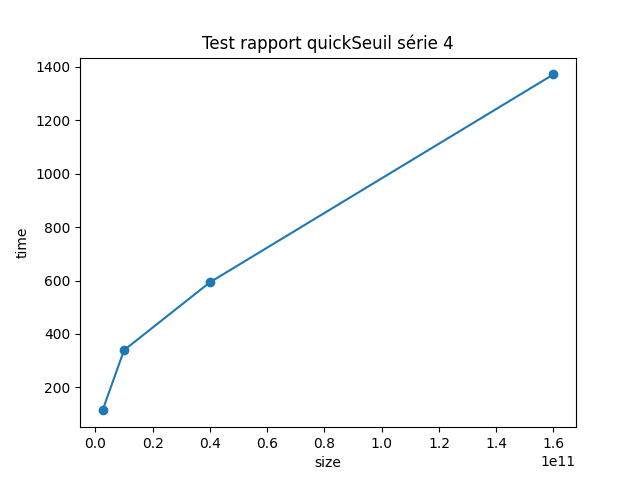
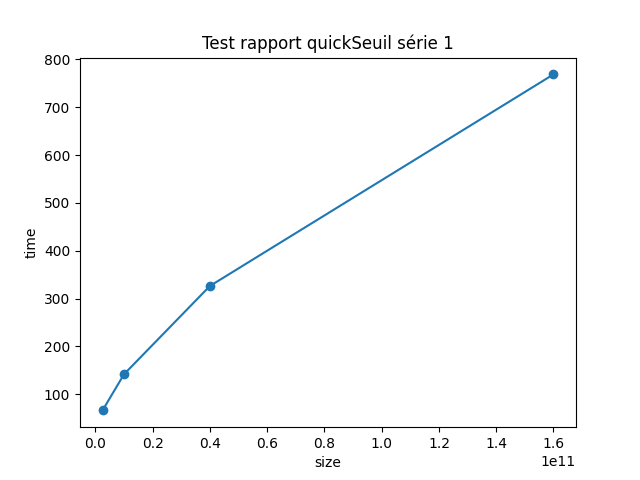
### Test du rapport

**

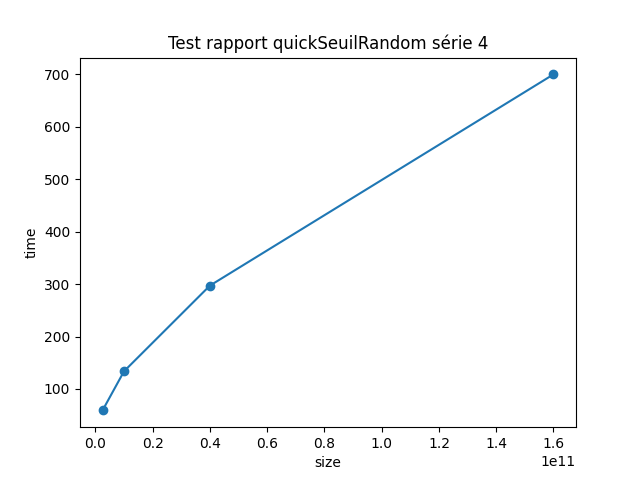
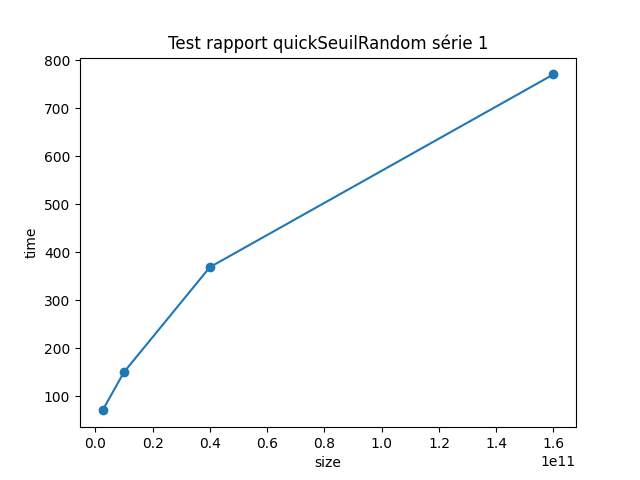
*Fig. 6.a, 6.b, 6.c (gauche à droite) : Tests du rapport pour tri par dénombrement*

**

*Fig. 7.a, 7.b (gauche à droite) : Tests du rapport pour tri rapide v1*

**

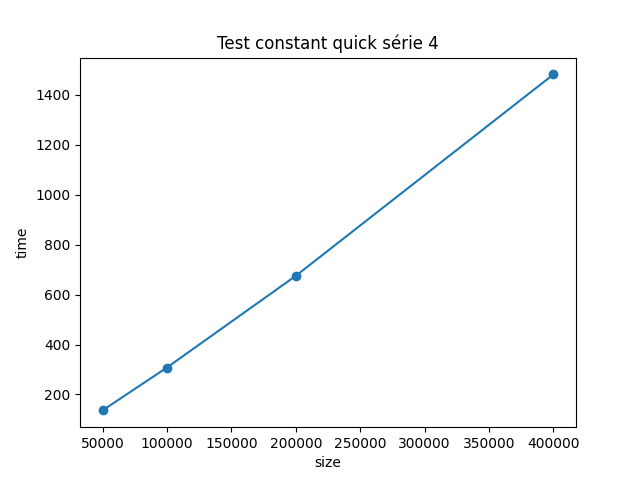
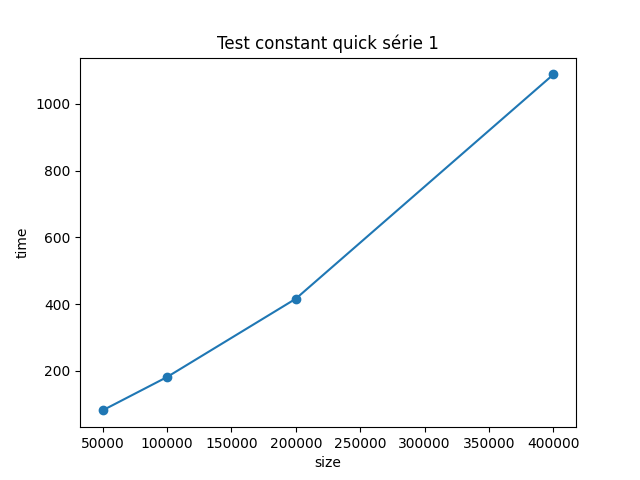
*Fig. 8.a, 8.b (gauche à droite) : Tests du rapport pour tri rapide v2*

**

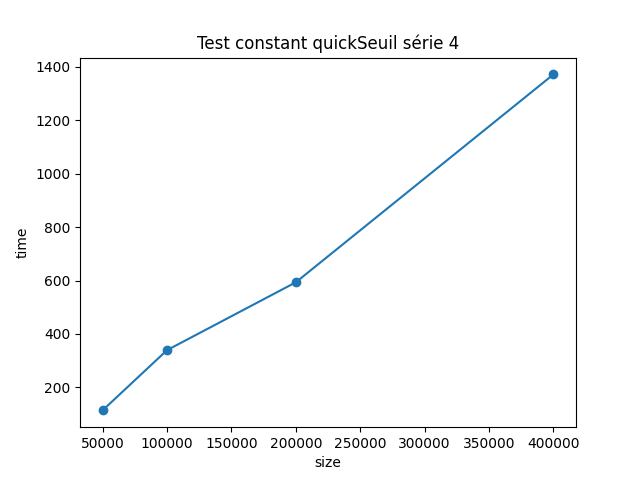
*Fig. 9.a, 9.b (gauche à droite) : Tests du rapport pour tri rapide v3*

### Test des constantes

*Fig. 10.a, 10.b, 10.c (gauche à droite) : Tests des constantes pour tri par dénombrement*

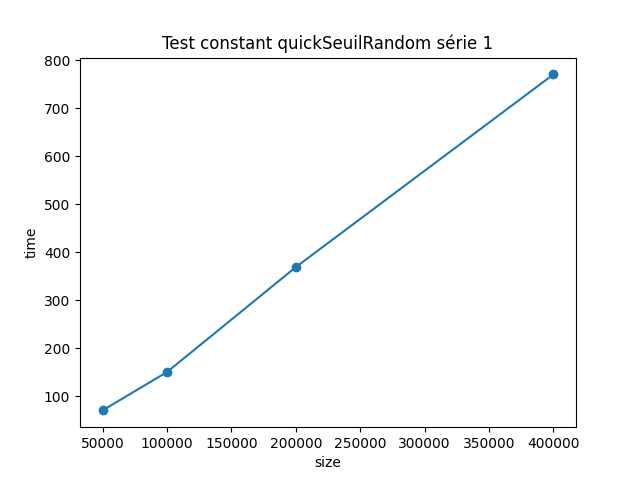
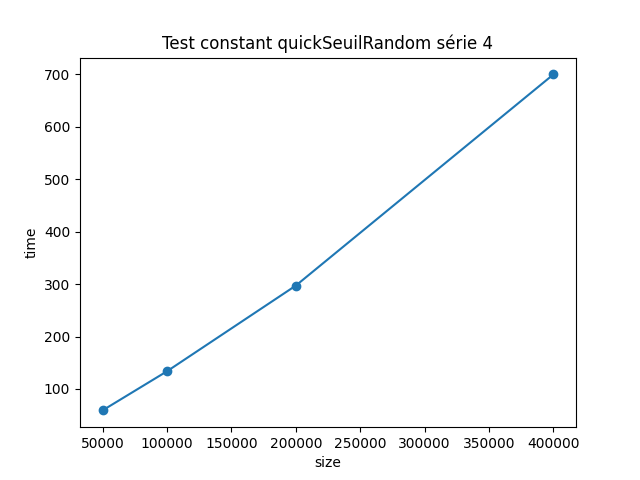
**

*Fig. 11.a, 11.b (gauche à droite) : Tests des constantes pour tri rapide v1*

*Une image contenant texte, ligne, capture d’écran, Tracé

Description générée automatiquement*

*Fig. 12.a, 12.b (gauche à droite) : Tests des constantes pour tri rapide v2*

**

*Fig. 13.a, 13.b (gauche à droite) : Tests des constantes pour tri rapide v3*

# Analyse et discussion

|  |  |
| --- | --- |
| 0 | / 6 pt |

### Que pouvez-vous déduire du test de puissance?

Pour tous les tests de puissance, on observe généralement ce qui s’apparente à une droite (un peu courbe pour counting série 2, un peu cassée pour quickSeuil série 4). Nous aurions donc affaire à une croissance polynomiale (autrement dit, un taux de croissance où m est la pente de la droite observée sur le test de puissance).

### Citez la consommation théorique en meilleur cas, en pire cas et en moyenne du temps de calcul pour les algorithmes, en notation asymptotique.

On note la taille et la magnitude des listes à trier.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Meilleur cas** | **Moyen cas** | **Pire cas** |
| **Counting sort** | \* | \* | \* |
| **Quicksort** |  |  |  |
| **Quicksort seuil** |  |  |  |
| **Quicksort seuil random** |  |  |  |

\* On pourrait optimiser l’algorithme de tri dénombrement que nous avons codé avec un tableau intermédiaire de taille au lieu de . Dans ce cas-là on remplacerait la magnitude par la plage dans les complexités.

### Que pouvez-vous déduire du test du rapport?

Malgré le manque de point, on observe une convergence. Les courbes semblent donc valider notre hypothèse selon laquelle le temps de calcul croît en temps quadratique par rapport à la taille des exemplaires.

### Que pouvez-vous déduire du test des constantes?

Les courbes des tests des constantes ont un comportement globalement affine, ce qui une nouvelle fois va de le sens d’une croissance en n².

### Discutez de l’impact du seuil de récursivité.

On constate que peu importe la série utilisée, l’utilisation du seuil de récursivité obtenu expérimentalement améliore les performances, avec un gain de l’ordre de quelques centaines de ms pour les séries avec beaucoup d'exemplaires.

Néanmoins, d’après les tests précédents, cela ne semble pas modifier drastiquement le comportement asymptotique.

### Suite à cette analyse, indiquez sous quelles conditions (taille d’exemplaire ou autre) vous utiliseriez chacun de ces algorithmes. Justifiez.

En faisant une synthèse de nos premières observations et nos tests, à faible magnitude et faible taille nous choisirions le tri par dénombrement (nous avons déjà évoqué qu’il était plus performant).

Dès que la magnitude ou la taille deviennent assez élevées, nous choisirions le tri rapide. Comme expliqué dans les premières observations, on préférera dans tous les cas choisir un pivot aléatoire (qui améliore la complexité dans le pire des cas - à savoir une liste presque triée). De nos observations et de ce que nous avons précédemment expliqué dans nos analyses, nous préférerons également imposer un seuil de récursivité (de l’ordre de 25).

# Autres critères de correction

### Respect de l’interface tp.sh

|  |  |
| --- | --- |
| 0 | / 1 pt |

Utilisation

tp.sh -a [counting | quick | quickSeuil | quickRandomSeuil] -e [path\_vers\_exemplaire]

Arguments optionnels

-p affiche les nombres triés en ordre croissant sur une ligne, sans texte superflu

-t affiche le temps d’exécution en ms, sans unité ni texte superflu

Important: l’option -e doit accepter des fichiers avec des paths absolus.

### Qualité du code

|  |  |
| --- | --- |
| 0 | / 1 pt |

### Présentation générale

|  |  |
| --- | --- |
| 0 | / 1 pt |

* Concision
* Qualité du français

### Pénalité retard

|  |
| --- |
| 0 |

* -1 pt / journée de retard, arrondi vers le haut. Les TPs ne sont plus acceptés après 3 jours.